Distributed Concept Drift Detection for Efficient Model Adaptation with Big Data Streams

Ian Whitehouse
Department of Computer Science
American University
Washington, DC, USA
iw9892a@american.edu

Rodrigo Yepez-Lopez
Department of Computer Science
American University
Washington, DC, USA
ry4789a@american.edu

Roberto Corizzo
Department of Computer Science
American University
Washington, DC, USA
rcorizzo@american.edu

***Abstract*—Predictive models are essential in big data platforms to tackle the needs of several real-world applications. However, static models are known to be prone to obsolescence in dynamic environments. While concept drift detection represents a viable way to deal with this problem, it is scarcely explored in the context of big data streams. In this paper, we propose a distributed drift detection workflow based on the DDM algorithm paired with a predictive model. Our workflow updates the predictive model as soon as drifts are detected, adjusting to the most recent data characteristics. To enable the analysis of large-scale datasets, we leverage Pandas UDFs and Apache Spark, efficiently distributing this workload across multiple worker node instances. Our experiments on two real-world drift detection datasets show the positive results obtained in terms of Speedup, Scaleup, and a limited impact in detection delay in comparison to a single worker node instance.**

***Index Terms*—Drift detection, predictive modeling, distributed systems, big data, Apache Spark.**

I. Introduction

The integration of predictive models in big data platforms and frameworks has shown promising results in domains characterized by large-scale data such as weather [1], [2], vehicular traffic [3], [4], and smart grids [5], [6]. However, a major pitfall of such methods is that, without taking concept drift detection into account, they are exposed to a number of scenarios that undermine their robustness in evolving environments. For instance, wrong predictions in stock prices due to concept drift may cause catastrophic losses in the financial domain [7] [8]. As drifts in the data distribution occur, it is important to keep predictive models updated, to prevent their rapid obsolescence.

Concept drift detection has become a recognized task in a variety of real-world applications characterized by dynamic environments with data streams, including cybersecurity, smart grids, and settings that generate sensor data. Recent works involve the refinement of concept drifts in modern settings such as multi-class classification, multi-label data streams, and ensembles. The work in [9] proposes a drift detection method for binary classification settings that checks whether data assigned to a given class occupies spaces considered relevant to the other class. To this end, data is mapped into a quadtree-based memory structure that provides knowledge about which class (label) is dominant in a given region of the feature space. The authors in [10] propose an unsupervised concept drift detector leveraging dynamic ranking of temporal label dependencies and data fusion to support the analysis of multi-label data streams. Group concept drift for multiple data streams is addressed in [11], where an online learning algorithm using a distribution-free test statistic is proposed. A recent application of concept drift detection and deep neural model adaptation for next activity prediction in business data streams is proposed in [12].

The authors in [13] devise a semi-supervised drift detector based on an ensemble of classifiers with self-training and dynamic classifier selection. Ensemble-based online concept drift detection is also addressed in [14], where base classifiers are trained on random subsets of features to create a background ensemble that rapidly adapts to changes, adopting self-adjusting bagging to enhance the exposure of difficult instances from minority classes. The work in [15] proposes an active weighted aging ensemble algorithm to react to concept drift appropriately in an active learning setting with limited budget. Another emerging thread of research works is that of change detection and drift detection methods to support continual/lifelong learning models. This capability is particularly crucial in task-agnostic and task-free learning settings [14], [16]–[22].

Despite the progress completed in concept drift detection thanks to recent works on the subject, most of the research thus far does not take into account big data streams and does not tackle distributed learning capabilities of the underlying algorithms' approaches. As a result, the current solutions are not scalable, and appear inadequate for the monitoring and analysis of large-scale data leveraging big data frameworks.

In this paper, we attempt to fill this gap by proposing a distributed learning workflow based on the DDM drift detection algorithm, implemented in Apache Spark. The devised workflow features the interaction of a predictive model and a drift detector, which monitors the predictive model's error rate and detects concept drifts. As a drift is detected, the predictive model is updated with the most recent available data. Our distributed learning strategy adopts the Apache Spark framework to perform model training and drift detection in parallel on multiple worker nodes, allowing us to efficiently

979-8-3503-2445-7/23/$31.00 ©2023 IEEE

analyze large-scale data streams. Our experiments with two real-world datasets show the merit of the proposed solution in terms of its detection delay and its scalability when compared to single executor counterpart.

The paper presents the following structure: Section II describes our proposed approach in detail. Section III outlines the analyzed datasets, the experimental setup, and discusses the results of our experiments. Finally, Section IV wraps up the paper by providing possible directions for future work.

II. Method

In this section, we describe our proposed distributed drift detection approach. Drift detection with predictive modeling, and distributed scheduling are described in separate subsections. An overview of the proposed workflow is shown in Figures 1 and 2.

A. Drift detection with predictive modeling

DDM (Drift Detection Method) [23] is a concept drift detection method that analyzes the error rate of a predictive algorithm over time. It is based on the PAC learning model premise, which assumes the error rate of a predictive algorithm will decrease as the number of analysed samples increase, as long as the data distribution is stationary. If the algorithm detects an increase in the error rate that goes beyond a computed threshold, two options are possible: a drift is detected, or the algorithm warns the user that a drift may occur in the near future, which is known as the warning zone.

Considering a data stream, where s_{min} is the minimum recorded standard deviation thus far, the detection algorithm leverages the following time-variant information:

- p_i: The error rate at time point i.
- s_i: The standard deviation at time point i.

The detection threshold is then computed as a function of two statistics, obtained when (p_i+s_i) is the minimum recorded error rate: The conditions for entering the warning zone can be formalized as:

$$p_i + s_i \geq p_{min} + 2 * s_{min}.$$

Similarly, the condition that triggers the detection of a concept drift can be formalized as:

$$p_i + s_i \geq p_{min} + 3 * s_{min}.$$

DDM requires a predictive model for the continuous analysis of its error rate. To this end, in our work, we adopt the Random Forest classifier [24]. The rationale for its adoption is based on the high potential that ensemble-based and combination methods based on hybrid models can provide in contexts with complex data characteristics, which naturally occur in real-world data streams [8]. In such contexts, model ensembles such as Random Forest generally provide better and more consistent results than single models such as Support Vector Machines [25]. Specifically, Random Forest is composed of multiple decision trees that capture complex nonlinear relationships in the data, and it provides native mechanisms to deal with overfitting, such as random feature selection and bootstrap sampling. Additionally, it provides high robustness to noisy data and outliers, which occur in real world data. Another aspect worth consideration is that, since each tree in the ensemble is independent, their training process can be run in a multi-threaded fashion, leading to faster training phase and improved computational efficiency, which is a salient aspect in our workflow.

To implement our workflow, we leverage the DDM implementation in *scikit-multiflow*[1] and the Random Forest classifier implementation in *scikit-learn*[2].

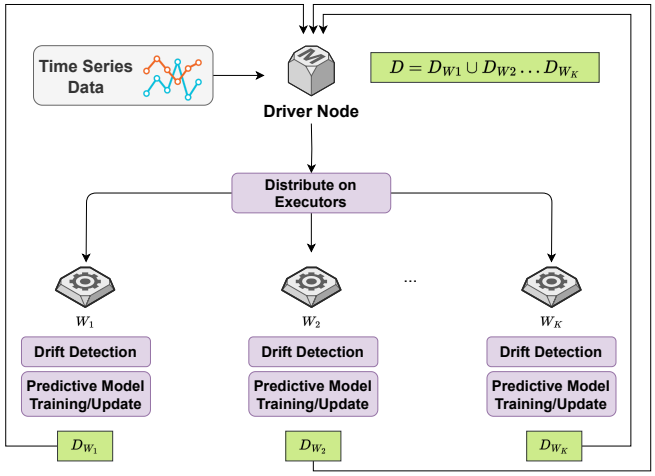


Fig. 1. Graphical overview of the proposed approach for distributed drift detection. A Spark driver node distributes data across different worker node instances, where drift detection and predictive model's training/update stages occur. Partial results at each worker are returned to the driver, where the complete set of drifts is obtained via aggregation.

B. Distributed scheduling

Our distributed workflow implementation leverages Pandas User-Defined Function (Pandas UDFs) and the Apache Spark framework to enable distributed computation [26]. Pandas UDFs, also referred to as vectorized UDFs, are optimized functions that utilize Apache Arrow for efficient data transfer, enabling vectorized operations. This backing data format allows Pandas UDFs to enhance performance by up to $100x$ when compared to regular Python UDFs, which can only perform tasks on a row-by-row basis. Specifically, in our framework, the DDM drift detection and predictive model training and update are implemented as a Pandas UDF, adopting iterators on Pandas series.

Our strategy loads the time series dataset in memory as a distributed Spark DataFrame via PySpark, i.e. the Python Apache Spark API. The driver node, which is in charge of orchestrating the computational workload, initializes the number of partitions based on the cluster configuration, consisting of the number of worker node instances/executors, the number

[1]https://scikit-multiflow.readthedocs.io/en/stable/api/generated/skmultiflow.drift_detection.DDM.html?highlight=ddm

[2]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

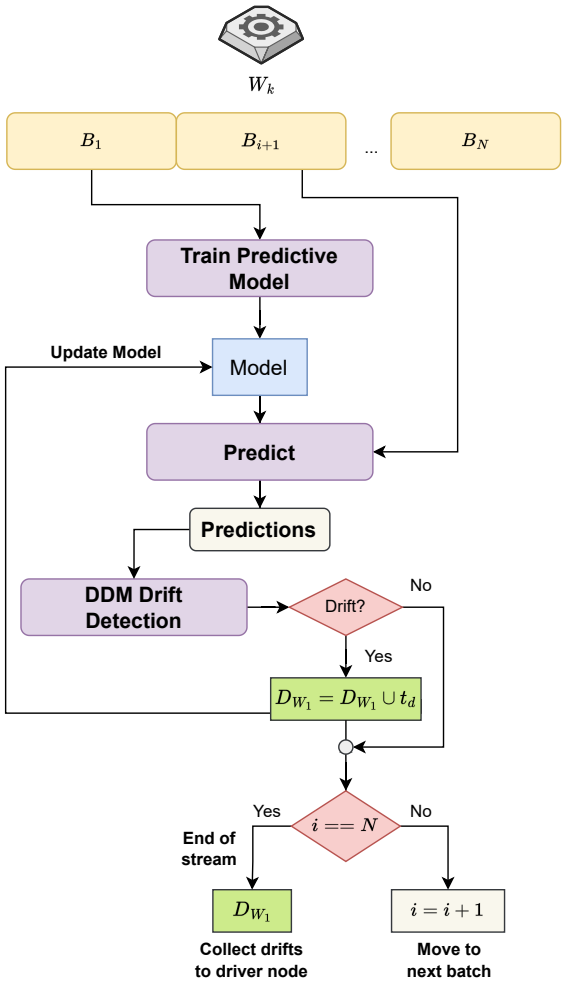


Fig. 2. Single worker node instance's perspective of the distributed drift detection workflow. Batch-oriented processing takes place, where the DDM drift detection algorithm monitors the error rate of a predictive model, triggering model updates as soon as a drift is detected. At the end of the stream, the worker node instance returns the detected drifts to the driver node.

of cores for each instance, and the amount of RAM memory for each instance. The driver nodes divides the DataFrame into partitions and assigns them to different worker node instances/executors available in the computational cluster.

A Python interpreter runs on each worker node instance on its partition, using all available local cores to run iterative operations on the Pandas DataFrame. A single worker node instance's perspective is graphically presented in Figure 2.

Each worker node is responsible for building and training a separate DDM and Random Forest models. Once a partition has been processed by a worker node instance, the set of drifts detected by each worker node on the local partition is returned to the driver, which collects and aggregates all partial results, leading to the final result. The pseudo-code of our workflow is described in Algorithm 1.

Result: Detected drifts D
Input: Time Series Data X
Distribute $X = \{B_1, B_2, \ldots, B_N\}$ to all workers
foreach *worker* $W_k \in \{1, 2, \ldots, K\}$ **do** in parallel
  $D_{W_k} \leftarrow \{\emptyset\}$ Local drifts
  $m \leftarrow$ train predictive model on $B_1 \leftarrow X[1]$
  foreach $i \in \{2, \ldots, N\}$ **do**
    $B_i \leftarrow X[i]$
    $\hat{y}_{B_i} \leftarrow m.predict(B_i)$
    $(drift, t_d) \leftarrow DDM(\hat{y}_{B_i})$
    if *drift* **then**
      $D_{W_k} \leftarrow D_{W_k} \cup t_d$
      $m \leftarrow m.fit(B_{i-1})$
    end
    if i==N **then**
      return D_{W_k}
    end
  end
end

Algorithm 1: Pseudo-code of the distributed workflow for model training and inference.

III. Experiments

In this section we provide a description of the datasets analyzed in our experiments, give details on our experimental setup, and analyze the results obtained in terms of drift detection delay and scalability. Our code implementation is accessible at: https://github.com/rcorizzo/distributed-drift-detection.

TABLE I
Datasets analyzed in our experiments. Resampled refers to the maximum replication factor considered (Outdoor Stream: 512, Rialto: 32).

Dataset	Instances (original)	Instances (resampled)	Features
Outdoor Stream	4.000	2.048.000	21
Rialto	82.250	2.632.000	27

A. Datasets

We analyze the following real-world concept drift detection datasets, also described in Table I:

- **Outdoor Stream** [27] contains images recorded by a mobile device in a garden environment. The task is to classify 40 different objects, each approached ten times under varying lighting conditions affecting the color-based representation. The objects are encoded in a normalized 21-dimensional RG-Chromaticity histogram.
- **Rialto** [28] contains ten of the colorful buildings next to the famous Rialto bridge in Venice, encoded in a normalized 27-dimensional RGB histogram. Images are obtained from time-lapse videos captured by a webcam with fixed position. The recordings cover 20 consecutive days during May-June 2016. Continuously changing weather and lighting conditions affect the representation.

B. Setup and Metrics

In our experiments, we are interested in measuring the effectiveness of our concept drift detection and predictive

TABLE II
EXECUTION TIMES WITH DIFFERENT CLUSTER CONFIGURATIONS (INSTANCES, CORES) FOR ALL ANALYZED DATASETS.

	Rialto dataset														
	1 Instance			**2 Instances**			**4 Instances**			**8 Instances**			**16 Instances**		
Factor	**2 cores**	**4 cores**	**8 cores**	**2 cores**	**4 cores**	**8 cores**	**2 cores**	**4 cores**	**8 cores**	**2 cores**	**4 cores**	**8 cores**	**2 cores**	**4 cores**	**8 cores**
1	30.45	31.40	34.44	19.88	22.03	24.49	19.81	22.03	26.84	26.85	35.71	47.48	38.69	50.16	54.48
2	48.61	49.15	54.29	29.16	31.46	35.17	28.30	32.03	34.13	21.72	27.19	34.18	33.34	47.73	56.78
4	80.99	82.79	92.59	46.34	49.34	54.97	45.62	50.18	53.11	30.64	36.33	44.76	26.77	35.28	48.25
8	152.14	153.48	169.32	81.93	85.04	95.43	79.49	85.35	89.38	48.33	54.25	62.61	35.47	44.31	58.33
16	289.61	293.48	328.29	154.86	154.94	176.87	123.89	158.61	160.30	82.55	90.75	97.86	54.50	63.42	77.46
32	572.22	573.62	645.27	293.97	299.87	333.92	283.97	302.45	302.59	151.64	162.98	169.13	89.51	99.89	113.99
	Outdoor Stream dataset														
	1 Instance			**2 Instances**			**4 Instances**			**8 Instances**			**16 Instances**		
Factor	**2 cores**	**4 cores**	**8 cores**	**2 cores**	**4 cores**	**8 cores**	**2 cores**	**4 cores**	**8 cores**	**2 cores**	**4 cores**	**8 cores**	**2 cores**	**4 cores**	**8 cores**
16	34.22	37.24	39.36	26.83	30.41	35.56	64.05	74.83	92.25	102.21	89.85	108.06	93.93	85.48	58.37
32	48.25	52.05	56.40	35.01	38.87	45.84	55.67	61.59	73.60	74.87	89.08	113.68	158.23	187.86	101.64
64	75.70	80.07	87.93	49.97	53.40	62.46	47.09	51.88	58.27	56.34	72.60	86.97	178.12	130.72	149.94
128	131.80	132.65	147.43	76.32	81.90	93.29	74.05	79.32	86.08	83.96	91.52	117.64	115.34	144.42	195.52
256	236.08	241.72	268.93	133.18	134.20	155.52	125.37	133.76	138.11	75.82	83.58	94.71	98.41	120.26	158.38
512	456.71	458.34	519.70	239.94	243.08	278.85	222.55	236.33	238.35	124.16	134.44	145.21	79.62	90.19	108.47

model update pipeline. To this end, we quantitatively measure the drift detection delay and the scalability of the workflow, comparing the execution time with a single worker node instance with multiple worker node instances. Our experiments involve multiple cluster configurations with $1, 2, 4, 8,$ and 16 worker node instances. To perform a fine-grained analysis of scalability, we also experiment with different configurations of the number of cores at each worker node instance $(2, 4, 8)$. Each instance is equipped with 8 GB of RAM memory. For scalability, we compute *Speedup* and *Scaleup* metrics. Specifically:

- *Speedup* is computed as the ratio between the time execution with the single-executor implementation, and the corresponding execution time with multiple worker node instances/executors.
- *Scaleup* is obtained by simultaneously increasing the problem size and the number of node instances/executors, and computing ratios between the execution time of any configuration with respect to the execution time with the initial configuration, i.e. simplest problem size and a single worker node instance/executor.

C. Results discussion

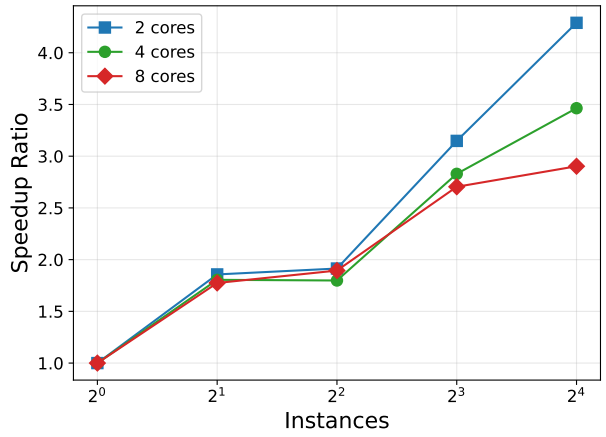
We performed our experiments on two real-world drift detection datasets (more information is reported in Table I) with a non-distributed execution (single worker node instance), and with a distributed execution ($2, 4, 8,$ and 16 worker node instances). Our scalability experiments are aimed at assessing the benefit of our distributed job scheduling workflow in terms of the reduction in execution time that can be achieved with multi-worker node instances. We remark that, the ideal curve for Speedup is a 45-degree line, whereas the ideal curve for Scaleup is a flat line ($y = 1$), which are rarely obtained in practice, due to computational and network bottlenecks that normally occur in distributed workflows.

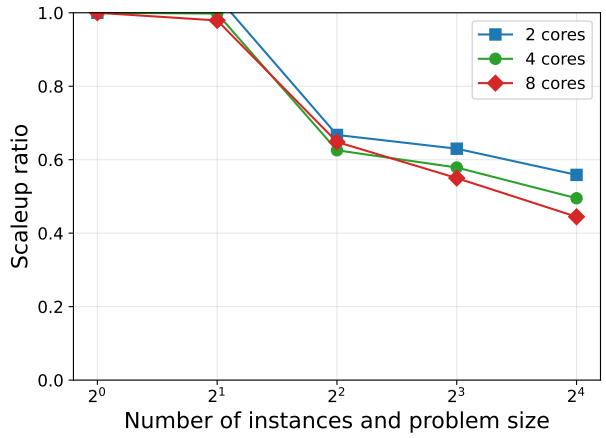
Experiments in Table II present the execution times obtained with different cluster configurations. We consider executions with multiple replication factors, where data rows are replicated to evaluate the effectiveness of our workflow on more challenging conditions with larger dataset sizes. For each replication factor, we run experiments 5 times and report averaged results. Figures 3 and 4 show the Speedup (top), Scaleup (center), and percentage of delay (bottom). Results in these figures are obtained considering a replication factor of $8x$ (Rialto) and $512x$ (Outdoor Stream).

By analyzing the execution times in Table II, we observe that the execution time increases as the number of cores on each executor increases from 2 to 4 and 8. This result suggests that larger-sized executors are not a good option for this specific workflow, which suffers from the overhead of data distribution. We also argue that this result may partially depend on the simplicity of the workflow, and that this outcome could change with the adoption of a more complex predictive model which fully leverages multi-threaded computation. However, comparing results with $1, 2, 4, 8,$ and 16 instances, highlights the merit of the distributed implementation. We observe a significant reduction in the execution time as more worker node instances are utilized in the majority of cases. This is particularly true with higher values of replication factor, i.e. when the data being analyzed is large enough to benefit from the distributed workflow.

Results in Figures 3 and 4 show that the Speedup factor increases linearly as the number of worker node instances is increased by a factor of 2. The maximum Speedup achieved is $4.28x$, with 16 executors (Rialto) and $5.73x$ with 16 executors (Outdoor Stream). One exception can be observed with 4 instances, where apparently no Speedup is achieved with respect to the configuration with 2 instances.

For Scaleup results we adopt different replication factors as problem sizes: $1, 2, 4, 8, 16$ (Rialto), $32, 64, 128, 256, 512$ (Outdoor Stream). The results depict a sub-optimal scenario where we observe decreasing values with an increasing number of worker node instances, largely from a visible drop from 2 to 4 instances. This result can be explained considering that, with problem size growth, the data transfer across worker node





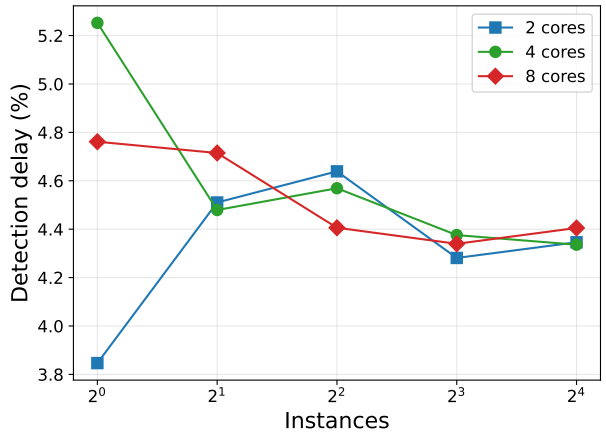
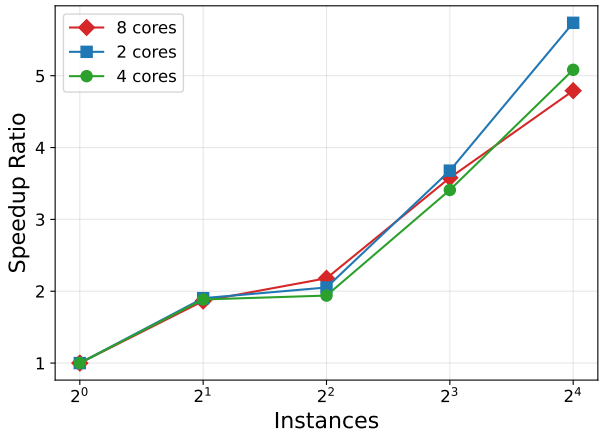
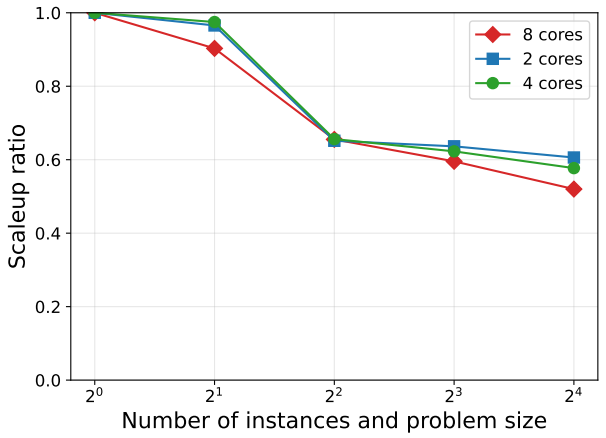


Fig. 3. Scalability results on Rialto dataset: Speedup (top), Scaleup (center), and percentage of delay (bottom).





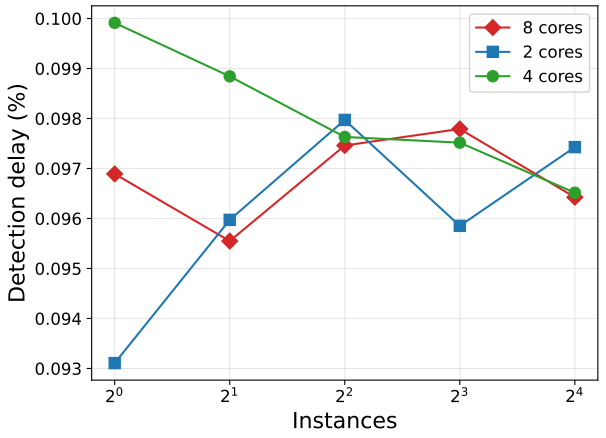


Fig. 4. Scalability results on Outdoor Stream dataset: Speedup (top), Scaleup (center), and percentage of delay (bottom).

instances also increases, leading to an overhead that factors in a higher execution time. However, Scaleup values still lie in a satisfactory range considering the challenges presented by the distributed workflow.

Results for detection delay show that the distributed nature of the workflow does not result in significant deviation between single and multi-worker instances. It can be observed that, for both datasets, detection delays converge to comparable values lying within a small range. Considering that each worker node instance performs drift detection and model training independently, the stability of our extracted results indicate that each worker node instance is provided with enough data to reliably compute distribution estimates for the DDM algorithm and to train an accurate Random Forest model. Consequently, the distributed workflow does not significantly alter the drift detection delay compared to its single-instance counterpart.

IV. Conclusion

In this paper, we proposed a distributed drift detection workflow for big data streams. We devised a strategy based on the DDM drift detection algorithm, paired with a Random Forest predictive model. Our approach adaptively updates the predictive model as soon as drifts are detected, preventing it from obsolescence. We leverage Pandas UDFs and Apache Spark to efficiently distribute the workload, partitioning the data across multiple worker node instances, which run the workflow in a parallel fashion. Our results on two real-world datasets show the effectiveness of our solution in terms of Speedup, Scaleup, and detection delay. As future work, we will investigate an extension of our approach to edge and federated learning settings. We will also analyze alternative drift detection approaches and more complex predictive models, and assess their suitability within our workflow.

Acknowledgment

The authors acknowledge the support of NVIDIA through the generous donation of GPUs in the context of the Data Science GPU Grant.

References

[1] J. Kalajdjieski, E. Zdravevski, R. Corizzo, P. Lameski, S. Kalajdziski, I. M. Pires, N. M. Garcia, and V. Trajkovik, "Air pollution prediction with multi-modal data and deep neural networks," *Remote Sensing*, vol. 12, no. 24, p. 4142, 2020.

[2] A. C. Onal, O. B. Sezer, M. Ozbayoglu, and E. Dogdu, "Weather data analysis and sensor fault detection using an extended iot framework with semantics, big data, and machine learning," in *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2017, pp. 2037–2046.

[3] I. Lana, J. Del Ser, M. Velez, and E. I. Vlahogianni, "Road traffic forecasting: Recent advances and new challenges," *IEEE Intelligent Transportation Systems Magazine*, vol. 10, no. 2, pp. 93–109, 2018.

[4] Y. Lv, Y. Duan, W. Kang, Z. Li, and F.-Y. Wang, "Traffic flow prediction with big data: a deep learning approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 865–873, 2014.

[5] A. A. Munshi and Y. A.-R. I. Mohamed, "Data lake lambda architecture for smart grids big data analytics," *IEEE Access*, vol. 6, pp. 40 463–40 471, 2018.

[6] M. Altieri, R. Corizzo, and M. Ceci, "Scalable forecasting in sensor networks with graph convolutional lstm models," in *2022 IEEE International Conference on Big Data (Big Data)*. IEEE, 2022, pp. 4595–4600.

[7] X. You, M. Zhang, D. Ding, F. Feng, and Y. Huang, "Learning to learn the future: Modeling concept drifts in time series prediction," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 2434–2443.

[8] R. Corizzo and J. Rosen, "Stock market prediction with time series data and news headlines: a stacking ensemble approach," *Journal of Intelligent Information Systems*, pp. 1–30, 2023.

[9] R. A. Coelho, L. C. B. Torres, and C. L. de Castro, "Concept drift detection with quadtree-based spatial mapping of streaming data," *Information Sciences*, vol. 625, pp. 578–592, 2023.

[10] E. B. Gulcan and F. Can, "Unsupervised concept drift detection for multi-label data streams," *Artificial Intelligence Review*, vol. 56, no. 3, pp. 2401–2434, 2023.

[11] H. Yu, W. Liu, J. Lu, Y. Wen, X. Luo, and G. Zhang, "Detecting group concept drift from multiple data streams," *Pattern Recognition*, vol. 134, p. 109113, 2023.

[12] V. Pasquadibisceglie, A. Appice, G. Castellano, and D. Malerba, "Darwin: An online deep learning approach to handle concept drifts in predictive process monitoring," *Engineering Applications of Artificial Intelligence*, vol. 123, p. 106461, 2023.

[13] F. Pinagé, E. M. dos Santos, and J. Gama, "A drift detection method based on dynamic classifier selection," *Data Mining and Knowledge Discovery*, vol. 34, pp. 50–74, 2020.

[14] A. Cano and B. Krawczyk, "Rose: robust online self-adjusting ensemble for continual learning on imbalanced drifting data streams," *Machine Learning*, vol. 111, no. 7, pp. 2561–2599, 2022.

[15] M. Woźniak, P. Zyblewski, and P. Ksieniewicz, "Active weighted aging ensemble for drifted data stream classification," *Information Sciences*, vol. 630, pp. 286–304, 2023.

[16] B. Krawczyk, "Tensor decision trees for continual learning from drifting data streams," *Machine Learning*, vol. 110, no. 11-12, pp. 3015–3035, 2021.

[17] K. Faber, R. Corizzo, B. Sniezynski, M. Baron, and N. Japkowicz, "Lifewatch: Lifelong wasserstein change point detection," in *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2022, pp. 1–8.

[18] L. Korycki and B. Krawczyk, "Class-incremental experience replay for continual learning under concept drift," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 3649–3658.

[19] R. Corizzo, M. Baron, and N. Japkowicz, "Cpdga: Change point driven growing auto-encoder for lifelong anomaly detection," *Knowledge-Based Systems*, vol. 247, p. 108756, 2022.

[20] S. Basterrech and M. Woźniak, "Tracking changes using kullback-leibler divergence for the continual learning," in *2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2022, pp. 3279–3285.

[21] M. Roseberry, B. Krawczyk, Y. Djenouri, and A. Cano, "Self-adjusting k nearest neighbors for continual learning from multi-label drifting data streams," *Neurocomputing*, vol. 442, pp. 10–25, 2021.

[22] K. Faber, R. Corizzo, B. Sniezynski, and N. Japkowicz, "Vlad: Task-agnostic vae-based lifelong anomaly detection," *Neural Networks*, 2023.

[23] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence–SBIA 2004: 17th Brazilian Symposium on Artificial Intelligence, Sao Luis, Maranhao, Brazil, September 29-Ocotber 1, 2004. Proceedings 17*. Springer, 2004, pp. 286–295.

[24] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001.

[25] M. Petković, M. Ceci, G. Pio, B. Škrlj, K. Kersting, and S. Džeroski, "Relational tree ensembles and feature rankings," *Knowledge-Based Systems*, vol. 251, p. 109254, 2022.

[26] R. Corizzo and T. Slenn, "Distributed node classification with graph attention networks," in *2022 IEEE International Conference on Big Data (Big Data)*. IEEE, 2022, pp. 3720–3725.

[27] V. Losing, B. Hammer, and H. Wersing, "Interactive online learning for obstacle classification on a mobile robot," in *2015 international joint conference on neural networks (ijcnn)*. IEEE, 2015, pp. 1–8.

[28] ——, "Knn classifier with self adjusting memory for heterogeneous concept drift," in *2016 IEEE 16th international conference on data mining (ICDM)*. IEEE, 2016, pp. 291–300.